# Bitwarden Security Assessment Report

ISSUE SUMMARIES, IMPACT ANALYSIS, AND RESOLUTION

BITWARDEN, INC

# Table of Contents

# Summary

In October 2021, Bitwarden engaged with cybersecurity firm Cure53 to perform penetration testing and a dedicated audit of the source code. A team of four dedicated testers from Cure53 were tasked with preparing and executing the audit. A total of nineteen days were invested to reach total coverage needed for Bitwarden.

Over the nineteen days, twenty-five issues were discovered with only one vulnerability deemed critical and requiring immediate action. The breakdown and resolution of the issues is as follows:

- 16 issues were fixed during the assessment
- 4 issues fixed post-assessment
- 1 low-risk issue accepted for usability
- 2 informational-only issues with upstream library issues
- 2 informational-only issues under planning and research

Given that a majority of the issues were immediately addressable and were remediated during the assessment, these results are very positive, especially considering the size and complexity of the code being examined. Any remaining issues were remediated post-assessment.

This report was prepared by the Bitwarden team to cover the scope and impact of the issues found during the assessment and their resolution steps. Some issues had no impact and were reported for information purposes only. These issues are not included in this report. For completeness and transparency, a copy of the report delivered by Cure53 has also been attached to this report.

# Issues

## BWN-02-003 WP1: HTML injection on SAML Single-Sign On

Bitwarden can act as a Service Provider for use with SAML-based Single Sign On (SSO) and allows enterprise organizations to configure their SSO Identity Provider (IdP) service URLs under the organization's settings. It was discovered that lack of validation could allow a malicious customer to configure a SAML service URL with malicious HTML. This malicious HTML could possibly be used to execute cross-site scripting attacks or phishing campaigns under a Bitwarden domain. The configured Content Security Policy prevented this issue from being exploitable.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1691
2. https://github.com/bitwarden/server/pull/1695

Additional checks were added on the SSO configuration APIs to validate the SAML service URLs to make sure that they do not contain HTML characters such as <, >, and quotes. We don't expect that real service URLs will ever need these characters. Additionally, validation was added to ensure that all service URLs start with http:// or https://, as would be expected.

## BWN-02-004 WP1: Internal IP detection can be bypassed via IPv6

Bitwarden operates an “Icon Service” that is used to dynamically fetch icons to display next to items in a user’s vault. Given the dynamic nature of this service’s input parameters, a malicious actor could pass certain domain names and IP addresses in an attempt to probe internal networks that the attacker would otherwise not have access to. Several checks were already in place to prevent such attacks, however, it was discovered that certain IPv6 structures could circumvent existing checks. Given that the Bitwarden Icon service is operated in isolation, successful network probing was not exploitable.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1701

The Bitwarden Icon Service was updated to include :ffff: from the ::ffff:<ipv4 address> syntax so that the validation check correctly includes IPv4 addresses represented in the IPv6 standard.

## BWN-02-007 WP4: DOMXSS/Open redirect in webauthn-connector.html

Bitwarden provides a “webauthn connector” that is hosted on the web vault domain to enable non-web based applications to easily invoke FIDO2 WebAuthn APIs via iframe. These APIs are used to support WebAuthn security keys for two-step login. Once the WebAuthn functions are complete, a callback URI is redirected to using document.location.replace. This callback URI was being passed into the connector as a dynamic parameter. This callback URI parameter could be modified by a malicious actor to execute JavaScript or redirect unsuspecting users from the Bitwarden web vault domain. The connector is hosted outside of the Angular web vault application, so any executions would not be able to compromise sensitive vault data.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/web/pull/1282

It was determined that callback URI did not need to be dynamically provided and has been scoped to static values.

## BWN-02-008 WP4: Iframe injection in duo-connector.html

Bitwarden provides a “duo connector” that is hosted on the web vault domain to enable non-web based applications to easily invoke the Duo Web SDK APIs via iframe. These APIs are used to support the Duo service for two-step login. The Duo Web SDK requires that a dynamic hostname be provided from the Duo application configuration. This hostname value is configured on the user or organization account settings within Bitwarden and is then passed to the Duo Web SDK through the duo connector. While this hostname value is validated server side when being configured in account settings, it is not re-validated client side in the duo connector. A malicious actor could invoke the duo connector using a malicious hostname value, which the Duo Web SDK would then attempt to load. This could allow a malicious actor to embed deceptive pages in the duo connector iframe under the Bitwarden web vault domain.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/web/pull/1283

Since Duo application hostname values are always subdomains under the duosecurity.com or duofederal.com base domains, validation was added when parsing the passed hostname parameter in the duo connector.

## BWN-02-009 WP4: DOMXSS/Open redirect in captcha-connector.html

Bitwarden provides a “captcha connector” that is hosted on the web vault domain to enable non-web based applications to easily invoke captcha library APIs via iframe. These APIs are used to support captcha challenges that protect vital endpoints that are often attacked by automated traffic. Once the captcha response is complete, a callback URI is redirected to using document.location.replace. This callback URI was being passed into the connector as a dynamic parameter. This callback URI parameter could be modified by a malicious actor to execute JavaScript or redirect unsuspecting users from the Bitwarden web vault domain. The connector is hosted outside of the Angular web vault application, so any executions would not be able to compromise sensitive vault data.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/web/pull/1284

It was determined that callback URI did not need to be dynamically provided and has been scoped to static values.

## BWN-02-010 WP1: Collection restriction not enforced for managers

Organization owners can add users and assign a role to limit user access to vault data within the Bitwarden application. In addition to role based permissions, there are individual granular permissions that include the ability to restrict access to certain collections. It was discovered that the manager role had full access to all collections, even when the associated user has been restricted to certain collections. Organizations expect permissions to follow the model of least privilege so the expectation is that any permissions with additional restrictions should supersede those of the default roles.

### Resolution

Status: Issue has been fixed post-assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1834

Manager permissions have been updated to ensure that access is limited to only assigned collections. The PUT request now includes a user validation check to validate what collections that user is assigned.

## BWN-02-012 WP1: Stored XSS in Admin panel

The Bitwarden System Administrator Portal is a backend office tool used by Bitwarden employees to manage billing and other customer support and sales related operations for the Bitwarden service. The portal is restricted at the network layer and is not available for public access. The portal gives certain employees access to manage organization and user accounts, including deleting those entities. The delete button contains a JavaScript confirmation alert that includes the dynamic organization name. The organization name is set by end users of the Bitwarden application without restriction. A malicious user could perform a cross-site scripting attack on the portal by setting an organization name with values that would be executed by the unsanitized content of the JavaScript confirmation alert.

### Resolution

Status: Issue was fixed during the assessment.

Pull Requests:

1. https://github.com/bitwarden/server/pull/1703

The dynamic organization name and user email values were removed from the JavaScript confirmation alerts that are executed when using the delete buttons in the Bitwarden System Administrator Portal.

## BWN-02-016 WP1: Client-side cipher password reprompt can be bypassed

Even with an unlocked vault, Bitwarden clients offer the ability to further protect access to view data in the user's vault by turning on the "master password reprompt" feature at the item level. It was found that the reprompt feature is only evaluated client-side. A malicious actor could bypass the restrictions of the feature by manipulating the client's vault database or retrieving and decrypting the vault data manually. This behavior was previously known and understood when the feature was originally designed and released.

### Resolution

Status: Accepted for usability.

Pull requests: n/a

The master password reprompt feature is built for users that want to add an additional layer of protection for particular items to prevent accidental auto-fill, reveal, copy, or other misclicks. It is not intended to add an additional layer of cryptographic protection. To bypass the reprompt, malicious actors would need access to the device with the vault already being in an unlocked state. Users can prevent this by setting shorter vault timeout intervals. This is also highlighted in our help documentation: https://bitwarden.com/help/managing-items/#protect-individual-items

## BWN-02-017 WP1: Organization admins can hijack user-accounts

Bitwarden provides enterprise organization admins with the ability to utilize the “master password reset” feature. The master password reset feature requires opt-in enrollment from the individual organization members so that the members are aware of the trust issues (and possible risks) associated with allowing the organization to utilize this feature with their Bitwarden account. Enterprise organizations can also use the Single Sign-on (SSO) feature to federate authentication through an existing Identity Provider (IdP) used by the organization. This IdP is often administered by the same admins controlling the Bitwarden organization.

It was discovered that a malicious organization admin could authenticate using SSO as a target user into Bitwarden using a configured IdP that they control. This would then return the required `bearer token` that could be used to enroll the targeted user in Bitwarden’s master password reset feature where the admin could reset a new master password for the user. Since the malicious organization admin would not have access to the existing master password used by the targeted user, the admin would not be able to derive an encryption key needed to view the user’s existing vault data. This action would essentially corrupt existing vault data. However, if the targeted user were then to set a new master password or rotate their encryption key themselves following this event (essentially fixing their corrupted account back into a useable state), the malicious organization admin could then further utilize the master password reset feature to access the targeted user’s Bitwarden account without restriction.

### Resolution

Status: Issue has been fixed post-assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1940
2. https://github.com/bitwarden/jslib/pull/810
3. https://github.com/bitwarden/clients/pull/2845
4. https://github.com/bitwarden/web/pull/1698

Flows have been updated so that users are prompted for their master password to confirm their enrollment in admin password reset for their organization. Users are additionally notified in the UI that organization administrators will have the ability to change their master password.

## BWN-02-018 WP1: Insufficient CSRF protection in Admin panel

The Bitwarden System Administrator Portal is a backend office tool used by Bitwarden employees to manage billing and other customer support and sales related operations for the Bitwarden service. The portal is restricted at the network layer and is not available for public access.

ASP.NET Core, the framework used to build the Bitwarden System Administrator Portal, provides an attribute called `ValidateAntiForgeryToken` that can be easily applied to page actions to prevent CSRF style attacks from occurring. Certain administrative actions within the portal were discovered to be missing the `ValidateAntiForgeryToken` attribute, which could make them vulnerable to execution by a CSRF attack.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1696

The `ValidateAntiForgeryToken` attribute has been applied to all functions where it was missing in the Bitwarden System Administrator Portal.

## BWN-02-024 WP1: Open redirect via SAML SSO

Bitwarden can act as a Service Provider for use with SAML-based Single Sign On (SSO) and allows enterprise organizations to configure their SSO Identity Provider (IdP) service URLs under the organization's settings. It was discovered that a malicious customer could configure their organization with a malicious SSO service URL which would create a static redirect endpoint under the Bitwarden domain.

### Resolution

Status: Issue has been fixed post-assessment.

Pull requests:

1. https://github.com/bitwarden/server/pull/1948
2. https://github.com/bitwarden/server/pull/2085
3. https://github.com/bitwarden/jslib/pull/780

A signed "SSO token" has been added to the relay state of the SSO login process that provides protections similar to a CSRF token. The token is retrieved by a Bitwarden client from the server as part of the first "pre-validate" step of a client-initiated SSO process. The token is signed by the server with a short expiration window and is passed back to the server from the client as part of relay state. Subsequent SSO steps, such as the "external challenge", validate the token before proceeding with service URL redirects. The dynamic makeup of the expiring SSO token prevents organization's from sharing a static endpoint under the Bitwarden domain that could redirect to malicious endpoints.

## BWN-02-011 WP4: DOMXSS via ssoHandOffMessage cookie

Bitwarden provides a “SSO connector” that is hosted under the web vault domain. The connector enables client applications that are not browser-based to authenticate with organizations using Single Sign-on (SSO) protocols that require a web browser. In the case of the SSO connector serving the browser extension client, a “hand off message” can be displayed on screen after the SSO process completes that directs the user back to the browser extension. This hand off message can be dynamically set by the SSO server via cookies and is read by the connector. The hand of message was not sanitized properly, which could allow a malicious actor to execute arbitrary HTML and/or JavaScript under the Bitwarden web vault domain. Bitwarden’s existing Content Security Policy (CSP) effectively blocked any possibility for JavaScript to be executed, however, HTML/DOM manipulation of the connector itself was still possible.

### Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/web/pull/1285

Proper sanitization of the hand off message was implemented by setting the `innerText` property of the DOM element (DIV), therefore preventing an arbitrary DOM manipulation or JavaScript execution.

## BWN-02-019 Web: General HTTP security headers missing

The Bitwarden service includes many different endpoints and APIs that power client applications. It was discovered that certain API endpoints (those returning JSON responses to clients) were not returning best-practice security headers that can help minimize attack surfaces.

HTML-facing endpoints, such as the web vault (vault.bitwarden.com), were already effectively returning security headers and were not vulnerable.

### Resolution

Status: Issue was fixed during the assessment.

Pull Requests:

1. https://github.com/bitwarden/server/pull/1700

Special security headers, such as X-Frame-Options, X-Content-Type-Options, X-XSS-Protection, and Strict-Transport-Security, were added to all responses from Bitwarden API endpoints.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

## Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *BWN-02-001*) for the purpose of facilitating any future follow-up correspondence.

### BWN-02-003 WP1: HTML injection on SAML Single-Sign On *(Medium)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was found that the SAML2 library *Sustainsys.Saml2* used for the SAML SSO login does not properly sanitize parameters like the SAML2 IdP service URL before embedding them directly into HTML markup. This is done with *string.Format* when using the *POST* binding. This induces the risk of attackers injecting malicious markup and constructing a malicious site under a trusted Bitwarden domain.

The behavior could be abused in a Phishing attack where a victim is lured onto the trust-inducing URL. Here attackers could prompt them for user-credentials and, ultimately, steal them. A fully-blown Cross-Site Scripting vulnerability is prevented by modern browsers due to a strict Content Security Policy set by the library, which meant downgrading this issue to *Medium*.

**PoC:**
https://sso.bitwarden.com/Account/ExternalChallenge?scheme=dede72b9-821c-4221-863d-add00004b472

**Affected file:**
*Sustainsys.Saml2/WebSSO/Saml2PostBinding.cs*

**Affected code:**

```
public override CommandResult Bind<TMessage>(
       TMessage message, ILoggerAdapter logger, Action<TMessage, XDocument,
Saml2BindingType> xmlCreatedNotification)
{
       [...]
       var cr = new CommandResult()
       {
       ContentType = "text/html",
       Content = string.Format(
              CultureInfo.InvariantCulture,
              PostHtmlFormatString,
```

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
                message.DestinationUrl,
                relayStateHtml,
                message.MessageName,
                encodedXml)
        };
[...]
private const string PostHtmlFormatString = @"[...]<html
xmlns=""http://www.w3.org/1999/xhtml"" xml:lang=""en"">[...]
<meta http-equiv=""Content-Security-Policy"" content=""script-src 'sha256-
H3SVZBYrbqBt3ncrT/nNmOb6nwCjC12cPQzh5jnW4Y0='"">
</head>
<body>
[...]
<form action=""{0}"" method=""post""
name=""sustainsysSamlPostBindingSubmit"">[...]";
```

**Steps to reproduce:**

1. Go to an organization, set an SSO identifier and set up a SAML2 SSO login
2. Fill all required fields and supply the following value as SSO service URL:
   `https://cure53.de/"><div style="z-index: 999;position: absolute; width: 100&#x25;; height: 100&#x25;;background-color: red;"><center><h1>Please Enter Your Master Password</h1><input type=password name=pw><input type=submit value=Unlock></center></div>`
3. Use the organization-identifier as the value of the *scheme* parameter within the following URL: *https://sso.bitwarden.com/Account/ExternalChallenge?scheme=<orgId>*
4. This trust-inducing URL can now be sent to victims, with a theft of the supplied credentials happening there.

It is recommended to report the issue to the vendor who should fix the issue by escaping relevant HTML meta characters of the parameters before embedding them into the HTML markup. By doing so, this issue would be remediated correctly for all users of the library. As an additional precaution, it is advisable to verify that the scheme of the URL is *http* or *https*.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-004 WP1: Internal IP detection can be bypassed via IPv6 *(Low)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

The Bitwarden Icon component handles the fetching of website icons, which are then used and displayed by the client-side applications. As this component parses the targeted domain from a fully user-controlled HTTP path, multiple checks are deployed to ensure no internal system can be reached. If the checks failed, this could leak information about the deployed systems or open ports. Here it was discovered that the deployed logic to detect internal IPv6 addresses is incorrectly implemented and can be bypassed.

The IPv6 standard allows mapping IPv4 addresses via the *::ffff:<ipv4 address>* syntax. As shown below, the code is trying to detect certain IPv6 addresses but is failing to detect *"::ffff:"*. It is, therefore, possible to pass this check by configuring a domain whose DNS AAAA entry points to an internal system.

**Affected file:**
*server-rc/src/Icons/Services/IconFetchingService.cs*

**Affected code:**

```
public static bool IsInternal(IPAddress ip)
        {
           [...]
            var ipString = ip.ToString();
            if (ipString == "::1" || ipString == "::")
            {
                return true;
            }

            // IPv6
            if (ip.AddressFamily ==
System.Net.Sockets.AddressFamily.InterNetworkV6)
            {
                return ipString.StartsWith("fc") || ipString.StartsWith("fd") ||
                    ipString.StartsWith("fe") || ipString.StartsWith("ff");
            }
```

**Example DNS entry:**
*Test.com - ::ffff:10.0.0.1*

It is recommended to include a check for the aforementioned *"::ffff:"* IPv6 syntax to ensure no internal systems can be reached via the icon service.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-006 WP4: Client-side path traversal via missing variable validation *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment. Proper validation will be added to other paths in due course.*

During the assessment of the client-side web application, it was discovered that user-controlled path variables are not validated before they are utilized to construct a HTTP URL. By traversing the path up, it is possible to reach arbitrary API endpoints.

The impact of this issue was reduced as most paths immediately display the vault's lock page and it was not possible to abuse this behavior further.

**Affected file:**
*web-rc/src/app/send/access.component.ts*

**Affected code:**

```
ngOnInit() {
        this.route.params.subscribe(async params => {
            this.id = params.sendId;
            this.key = params.key;
            if (this.key == null || this.id == null) {
                return;
            }
            await this.load();
        });
    }
 [...]
 async load() {
 [...]
        try {
            let sendResponse: SendAccessResponse = null;
            if (this.loading) {
                sendResponse = await this.apiService.postSendAccess(this.id,
this.accessRequest);
            } else {
                this.formPromise = this.apiService.postSendAccess(this.id,
this.accessRequest);
```

**Example URL:**
*https://vault.bitwarden.com/#/send/**GI5QdTPPj0eSx63UALSNfg%2F..%2F..%2F..%2F..%2Fcure53**/snklAeUVfOaq8j8vxSh4Pw*

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

**Sent HTTP request:**
*POST /cure53 HTTP/2*
*Host: vault.bitwarden.com*
*[...]*

It is recommended to validate user-controlled path variables before utilizing them to create a HTTP path. This ensures it is not possible to modify the targeted HTTP endpoint.

### BWN-02-007 WP4: DOMXSS/Open redirect in *webauthn-connector.html* (*High*)

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was discovered that a DOM-based XSS vulnerability exists in the *webauthn-connector.html* page. This page receives the string specified in the "*data*" parameter, which base64 decodes and parses as a JSON structure. The "*callbackUri*" property contained in this JSON is used as a redirect destination, however, the specified URL is not validated at all. Therefore, an attacker can perform XSS attacks by specifying the *javascript:* URL.

The issue can be reproduced by opening the following URL. If the PoC works correctly, an alert dialog will pop up.

**PoC (XSS):**
https://vault.bitwarden.com/webauthn-connector.html?parent=https://a&data=eyJjYWxsYmFja1VyaSI6ImphdmFzY3JpcHQ6YWxlcnQoZG9jdW1lbnQuZG9tYWluKS8vIn0%3D

**Decoded payload:**
```
{"callbackUri":"javascript:alert(document.domain)//"}
```

In addition, an attacker can use it as an open redirect by specifying any *http(s):* URL, effectively opening the possibilities for an attacker to conduct Phishing attacks. This would be done by redirecting unsuspecting visitors to a different website without them noticing.

The issue can be reproduced by opening the following URL. If the PoC works correctly, the navigation to https://example.com will happen.

**PoC for open redirect:**
https://vault.bitwarden.com/webauthn-connector.html?parent=https://a&data=eyJjYWxsYmFja1VyaSI6Imh0dHBzOi8vZXhhbXBsZS5jb20vIn0%3D

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The payload is received and used in the highlighted code.

**Affected file:**
*web-rc/src/connectors/webauthn.ts*

**Affected code:**
```
function parseParametersV2() {
    let dataObj: { data: any, headerText: string; btnText: string;
btnReturnText: string; callbackUri?: string } = null;
    try {
        dataObj = JSON.parse(b64Decode(getQsParam('data')));
    }
    catch (e) {
        error('Cannot parse data.');
        return;
    }

    callbackUri = dataObj.callbackUri;
    [...]
}
[...]
function error(message: string) {
    if (callbackUri) {
        document.location.replace(callbackUri + '?error=' +
             encodeURIComponent(message));
        returnButton(callbackUri + '?error=' + encodeURIComponent(message));
    } else {
        parent.postMessage('error|' + message, parentUrl);
    }
}
```

It is recommended to create an allow-list in which the trusted origins are specified. Navigation should only be possible for the listed origins.

### BWN-02-008 WP4: Iframe injection in *duo-connector.html* (High)

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was discovered that an arbitrary *https:* URL can be embedded in an iframe on the application's vault domain. The vulnerable *duo-connector.html* page receives the string specified in the "*host*" parameter and uses it as the host to be embedded in an iframe.

Since there is no host validation here, an arbitrary host can be embedded in the iframe. The iframe appears on the application's domain as if it were part of the content of a legitimate Bitwarden application. Thus, an attacker could perform Phishing attacks by displaying a fake login page. In addition, if the user has enabled the autofill of the

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Bitwarden WebExtension for the *vault.bitwarden.com* domain, the credentials can be stolen through "***BWN-01-001 Extension: Autofill only checks top-level domain (Medium)***", which was previously reported but accepted for improved usability.

The issue can be reproduced by opening the following URL. If the PoC works correctly, the https://example.com will be embedded in the iframe.

**PoC:**
https://vault.bitwarden.com/duo-connector.html?host=example.com%23&request=x:x

The affected code was found in the following code and is highlighted next.

**Affected file:**
*web-rc/src/connectors/duo.ts*

**Affected code:**
```
document.addEventListener('DOMContentLoaded', event => {
    const frameElement = document.createElement('iframe');
    frameElement.setAttribute('id', 'duo_iframe');
    setFrameHeight();
    document.body.appendChild(frameElement);

    const hostParam = getQsParam('host');
    const requestParam = getQsParam('request');
    DuoWebSDK.init({
        iframe: 'duo_iframe',
        host: hostParam,
        sig_request: requestParam,
        submit_callback: (form: any) => {
            invokeCSCode(form.elements.sig_response.value);
        },
    });
    [...]
});
```

It is recommended to create an allow-list in which the trusted origins are specified. Embedding should only be possible for the listed origins.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-009 WP4: DOMXSS/Open redirect in *captcha-connector.html* (*High*)

***Note:*** *The issue has been addressed and fixed during the assessment.*

Following the discovery of the BWN-02-007 issue, a similar XSS vulnerability was discovered in the *captcha-connector.html*.

The issue can be reproduced by opening the following URL and solving the CAPTCHA. If the PoC works correctly, the alert dialog will pop up.

**PoC (XSS):**
https://vault.bitwarden.com/captcha-connector.html?parent=https://vault.bitwarden.com&data=eyJjYWxsYmFja1VyaSI6ImphdmFzY3JpcHQ6YWxlcnQoZG9jdW1lbnQuZG9tYWluKS8vIiwic2l0ZUtleSI6ImJjMzhjOGEyLTUzMTEtNGU4Yy05ZGZjLTQ5ZTk5ZjZkZjQxNyJ9

**Decoded payload:**
```
{"callbackUri":"javascript:alert(document.domain)//","siteKey":"bc38c8a2-5311-4e8c-9dfc-49e99f6df417"}
```

In addition, an attacker can use it as an open redirect by specifying any *http(s):* URL, effectively opening the possibilities for an attacker to conduct Phishing attacks. This would be done by redirecting unsuspecting visitors to a different website without them noticing.

The issue can be reproduced by opening the following URL and solving the CAPTCHA. If the PoC works correctly, the navigation to https://example.com will happen.

**PoC for open redirect:**
https://vault.bitwarden.com/captcha-connector.html?parent=https://vault.bitwarden.com&data=eyJjYWxsYmFja1VyaSI6Imh0dHBzOi8vZXhhbXBsZS5jb20vIiwic2l0ZUtleSI6ImJjMzhjOGEyLTUzMTEtNGU4Yy05ZGZjLTQ5ZTk5ZjZkZjQxNyJ9

The payload is received and used in the highlighted code.

**Affected file:**
*web-rc/src/connectors/captcha.ts*

**Affected code:**
```
async function start() {
    sentSuccess = false;

    const data = getQsParam('data');
```

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
    [...]
    try {
        decodedData = JSON.parse(b64Decode(data));
    }
    catch (e) {
        error('Cannot parse data.');
        return;
    }
    callbackUri = decodedData.callbackUri;
    [...]
}

function captchaSuccess(response: string) {
    if (callbackUri) {
        document.location.replace(callbackUri + '?token=' +
encodeURIComponent(response));
    } else {
        success(response);
    }
}
```

It is recommended to create an allow-list in which the trusted origins are specified. Navigation should only be possible for the listed origins.

### BWN-02-010 WP1: Collection restriction not enforced for managers *(Medium)*

Organization owners have the capacity to not only add additional users but also to assign different permissions or roles to restrict their access. The deployed design includes restricting access to only certain collections as well. It was discovered that the manager role has full access to all collections, even when the associated user is restricted to only a certain collection.

The GUI will display all collections present but will not display the available configuration options in case the access has been restricted. This can be simply bypassed by sending the corresponding requests directly to the backend. The behavior is introduced by the *EditAssignedCollections* function, which allows the manager role to access any collection.

**Steps to reproduce:**

1. In an organization, add a user as a manager but restrict the access to only one collection.
2. Log in as the manager user.
3. Add an additional user to the allowed collection but intercept the request.
4. Modify the collection's ID to a collection the user should not have access to.
5. The request is accepted.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

**Example request: adding user to a collection:**

```
PUT /api/organizations/2742708a-cd00-4b73-8f06-add400cd1656/collections/
3fa95140-52ee-4694-b368-add500f13de4/users HTTP/2
Host: vault.bitwarden.com
[...]

[{"id":"d6787bd0-3caa-4eca-ad7b-
add500aa6125","readOnly":false,"hidePasswords":false}]

HTTP/2 200 OK
```

**Affected file:**
*Core/Context/CurrentContext.cs*

**Affected code:**

```
public async Task<bool> EditAssignedCollections(Guid orgId)
        {
            return await OrganizationManager(orgId) || (Organizations?.Any(
             o => o.Id == orgId
                       && (o.Permissions?.EditAssignedCollections ?? false))
                          ?? false);
        }
```

It is recommended to modify the backend's logic to ensure the restriction of collections is correctly enforced. In case a manager role should always have access to all collections, the GUI permission dialog should display this information.

## BWN-02-012 WP1: Stored XSS in Admin panel *(**Critical**)*

***Note:** The issue has been addressed and fixed during the assessment.*

It was found that the organization name and the email address were insufficiently sanitized before being embedded into a JavaScript event handler. The HTML sanitizer only encodes relevant markup characters to prevent escaping from the HTML attribute and does not further interpret the underlying context of the attribute. It was confirmed that this allows attackers to execute JavaScript in the browser of an authenticated Bitwarden administrator. In effect, it means granting attackers full access to the Bitwarden Administrator panel.

**Steps to reproduce:**

1. Register a free organization with the organization name of `'-window["ale"+"rt"](1)-'`
2. View the organization in the Administrator's panel and click on *Delete*

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3. JavaScript is executed; an alert message pops up

**Affected file:**
*src/Admin/Views/Organizations/View.cshtml*

**Affected code:**
```
<form asp-action="Delete" asp-route-id="@Model.Organization.Id"
      onsubmit="return confirm('Are you sure you want to delete this
organization (@Model.Organization.Name)?')">
```

**Also affected:**
*src/Admin/Views/Organizations/Index.cshtml*
*src/Admin/Views/Users/View.cshtml*
*src/Admin/Views/Users/Index.cshtml*

It is recommended that user-input is always sanitized with respect to the JavaScript and HTML context before being embedded into the HTTP response. By doing so, relevant characters are either escaped or removed from the payload. As they would be required for attackers to escape from the quoted JavaScript string literal, the approach would-mitigate this vulnerability.

### BWN-02-016 WP1: Client-side cipher password reprompt can be bypassed *(Low)*

***Note:*** *The issue is known by Bitwarden but was accepted for usability purposes*

It was found that the password reprompt set on ciphers only acted client-wise. Thus, it can simply be bypassed either by updating the cipher or retrieving and decrypting the data manually. This signifies the risk of users relying on this feature, albeit it does not grant additional security. Attackers that have access to a user's unlocked vault without knowing the password could therefore access hypersensitive cipher-data that was protected by the additional password-reprompt.

**Steps to reproduce:**

1. Use the web client and set the master password reprompt flag on a cipher
2. When accessing the cipher and being prompted for the password, execute the following JavaScript (F12->Developer Console):

   ```
   bitwardenContainerService.cryptoService.compareAndUpdateKeyHash = _ => 1;
   ```

3. Enter any password and click *Ok*
4. Access to the decrypted cipher should be granted

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

It is recommended to require a valid master password hash on the server-side before allowing access to the cipher data for ciphers that have the password reprompt flag set. In order for this to work, it is required that ciphers that have this flag set are not cached, making attackers query the API. With a revised approach, even attackers who have gained unauthorized access to an unlocked vault cannot retrieve or update the ciphers, preventing their decryption.

### BWN-02-017 WP1: Organization admins can hijack user-accounts *(Medium)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was found that organization admins can abuse the SSO login feature to log in as any other user who does not have a second factor installed. This can be subsequently used to enroll the user within the password-reset functionality without their consent in order to override their master password and fully lock the user out of their account.

Alternatively, they can wait for the user to change their password with the "*rotate my encryption key*" flag set, which will give the organization administrator access to their encryption key. As such, attackers can decrypt and access all of the victim's private and organization vaults.

**Attack scheme:**

1. Malicious organization administrator creates a SAML SSO configuration and specifies an IdP certificate with access to the associated private key
2. The victim - a Bitwarden user with plenty of private and personal vault items - is invited into the organization.
3. The user accepts the invitation. Note that the auto-enrollment policy for the password reset is disabled, resulting in no warning or messages being displayed to the user.
4. The organization administrator can now log in as the user by forging a valid SAML assertion with the public key of the user and sending it to the Bitwarden service provider.
5. Being authenticated as the victim, the attackers can now delete all ciphers that the user has access to, including ciphers of other organizations.
6. In order to decrypt the cipher contents, the attackers will enroll the user in the organization's password-reset feature[1] which does not require a valid master password.
7. If the user changes their own password with the *"rotate my encryption key"* option set, the client will automatically re-enroll the user within the password-reset

[1] https://bitwarden.com/help/article/admin-reset/#what-is-admin-password-reset

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

silently. The correct decryption key for all personal ciphers will be sent to the organization admin.
8. The organization's admin can now reset the password of the user and has full access to all of the user's decrypted private vaults and organizations.

It is recommended to reprompt for the password and verify it on the server-side when enrolling into the password-reset functionality or when *upserting* an Emergency Access. This should be done in a manner similar to how it is done with the email change. By doing so, attackers with vault access will be prevented from enrolling the password-reset functionality without knowing the password.

Additionally, the users should be warned about the SSO hijack functionality before joining the organization. It could also be considered to require that the user actively opts-in to the SSO authentication against their account. With that, organization administrators would not be obtaining access to the users' vaults without their explicit consent.

### BWN-02-018 WP1: Insufficient CSRF protection in Admin panel *(Medium)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

Several instances of a missing *ValidateAntiForgeryToken* attribute were found in the Admin panel on actions like *PromoteAdmin* or *CreateTransaction*. The *SameSite=Lax* cookie property can successfully be disarmed by sending the request from a trusted Bitwarden domain with the help of issues such as BWN-02-003, BWN-02-007 or BWN-02-009. This could be abused by attackers to perform a handful of actions, including the promotion of organization administrators or the creation of transactions.

**Affected file:**
*src/Admin/Controllers/ToolsController.cs*

**Affected code:**
```
[HttpPost]
public async Task<IActionResult> PromoteAdmin(PromoteAdminModel model) {
```

It is recommended to identify all relevant modifying actions within the Admin panel and supplement them with the *ValidateAntiForgeryToken* attribute. By doing so, CSRF attacks are stopped by the CSRF token verification which is automatically added prior to the action. The change would prevent this attack due to the lack of the abusers' knowledge of the anti-CSRF token.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-024 WP1: Open redirect via SAML SSO *(Low)*

It was found that the web application redirects to an arbitrary attacker-controlled location when visiting the XYZ SSO endpoint. This allows malicious attackers to lure victims into a trust-inducing Bitwarden domain that inadvertently redirects to a malicious page and exposes unaware users to Phishing attacks. This attack-vector is a weaker form of BWN-02-003.

**Steps to reproduce:**

1. Set up a valid SAML SSO configuration for an organization.
2. Supply a malicious redirect URL as the SSO service URL and set the SAML binding to *POST* or *Redirect*.
3. Lure the victim to a trust-inducing URL:, for instance: https://sso.bitwarden.com/Account/ExternalChallenge?scheme=dede72b9-821c-4221-863d-add00004b472
4. The victim is being redirected to the malicious redirect URL from *Step 2*.

This type of attack is usually mitigated by advertising the redirect to the user with a notice about leaving the Bitwarden domain. As this is not easily done for the SAML POST binding without altering the *Sustainsys* library, it could be a solution to require an anti-CSRF token and the *POST* method before invoking the SAML library. It is advisable to perform this in such a way that the browser of a victim lured onto a static URL does not perform the request due to an invalid CSRF token. Instead, the SAML binding should only be invoked after the user enters the organization identifier on the SSO login view and clicks on *login*.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

## Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

### BWN-02-001 WP2: Message handler does not apply URL encoding *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

The Bitwarden WebExtension deploys a content script, which forwards any *message* event that can be triggered by the currently loaded website, to the background script for further processing. It was discovered that in case the *authResult* command is sent, the background script does not apply any further validation or encoding to the additional parameters before including them in the *BrowserApi.createNewTab* call.

This allows to specify additional path parameters, which could potentially be abused to exploit the WebExtensions popup page. It must be noted that the current logic verifies that the message object was sent by the configured vault domain and no vulnerability was discovered in the popups' index page.

**Affected code:**
*browser-rc/src/content/message_handler.ts*

**Affected code:**

```
window.addEventListener('message', event => {
[...]
if (event.data.command && (event.data.command === 'authResult')) {
        chrome.runtime.sendMessage({
            command: event.data.command,
            code: event.data.code,
            state: event.data.state,
            referrer: event.source.location.hostname,
        });
    }
```

**Affected file:**
*background/runtime.background.ts*

**Affected code:**

```
case 'authResult':
const vaultUrl = this.environmentService.getWebVaultUrl();
```

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
if (msg.referrer == null || Utils.getHostname(vaultUrl) !== msg.referrer) {
return;
}

try {
BrowserApi.createNewTab('popup/index.html?uilocation=popout#/sso?code=' +
msg.code + '&state=' + msg.state);
}
```

Although this issue is not exploitable, it is recommended to at least URL-encode the user=controlled variables, especially as this is already implemented for the *'webAuthResult'* command, which is also exposed via the same content script.

### BWN-02-002 WP5: Overly permissive domain parsing *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

The WebExtensions password storage logic is fully relying on the top URL currently loaded. To ensure that the domain is correctly parsed, as it is used to retrieve the stored credentials, certain checks are deployed. It was discovered that currently a block-list approach is implemented for protocols, which only contains the *data:* protocol handler. This could introduce security issues, for instance via the *about:blank* page or similar pseudo protocols added to browsers.

As an example, the Safari web browsers allows adding arbitrary characters to the *about:blank* URL. In the end, this behavior was not exploitable as this page origin could not be accessed by another page and no iframe could be injected to leak credentials of any stored domain.

**Example input:**
*about:blank@test.com*

**Returned domain:**
*test.com*

**Affected file:**
*jslib-rc/common/src/misc/utils.ts*

**Affected code:**

```
static tldEndingRegex = /.*\.(com|net|org|edu|uk|gov|ca|de|jp|fr|au|ru|ch|io|es|
us|co|xyz|info|ly|mil)$/;
[...]
static getDomain(uriString: string): string {
```

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
if (uriString == null) {
    return null;
}

uriString = uriString.trim();
if (uriString === '') {
    return null;
}

if (uriString.startsWith('data:')) {
    return null;
}

let httpUrl = uriString.startsWith('http://') ||
     uriString.startsWith('https://');
if (!httpUrl && uriString.indexOf('://') < 0 &&
     Utils.tldEndingRegex.test(uriString)) {
    uriString = 'http://' + uriString;
    httpUrl = true;
}
```

It is recommended to modify the logic of the *getDomain* function. Bitwarden should verify that in case a protocol is specified, it exclusively contains supported protocols. In case this is not feasible, it should be taken into consideration to check the domain string for the "@" character, as this character should not be present in a specified domain string before appending it to the *http://* protocol.

### BWN-02-005 WP1: Potential DoS in Icon service via HTTP response size *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

The Icon service is using the DotNet HttpClient to retrieve the specified domain's index page as well as the *pages* icon. It was discovered that the default maximum response size of 2GB is used. As the returned index page is parsed into an HTML DOM structure, which consumes additional memory resources, an attacker could abuse this behavior to cause a Denial-of-Service on the server.

**MSDN documentation:**
*"The maximum number of bytes to buffer when reading the response content. The default value for this property is 2 gigabytes."*[2]

**Affected file:**
*server-rc/src/Icons/Services/IconFetchingService.cs*

[2] https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.maxrespon...size?view=net-5.0

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

**Affected code:**

```
_httpClient = new HttpClient(new HttpClientHandler
            {
                AllowAutoRedirect = false,
                AutomaticDecompression = DecompressionMethods.GZip |
                    DecompressionMethods.Deflate,
            });
            _httpClient.Timeout = TimeSpan.FromSeconds(20);
```

Given that neither an icon nor an HTML index page should require 2GB, it should be taken into consideration to reduce the maximum body length for responses. This would ensure that an attacker cannot easily cause a DoS by consuming all available memory.

### BWN-02-011 WP4: DOMXSS via *ssoHandOffMessage* cookie *(Low)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was discovered that a DOM-based XSS vulnerability via a cookie exists in the *sso-connector.html* page. The page receives the "*ssoHandOffMessage*" cookie's value and displays it via the *innerHTML* property. Since critical characters (e.g. "<",">") are not encoded at all, arbitrary HTML can be rendered if an attacker can set the "*ssoHandOffMessage*" cookie, for example, via a subdomain's XSS. Due to the CSP set on this page, Cure53 was unable to execute JavaScript via this issue, however it is still possible to perform Phishing attacks by displaying a fake login form.

**Steps to reproduce:**

1. Navigate to https://bitwarden.com/.
2. Assuming that an attacker found a way to set the "*ssoHandOffMessage*" cookie, set the cookie by executing the following JavaScript code on the browser's DevTools.

   ```
   document.cookie="ssoHandOffMessage=<s>test</
   s>;Path=/;Domain=bitwarden.com"
   ```

3. Navigate to https://vault.bitwarden.com/sso-connector.html?code=123&state=:clientId=browser. As a result of the *innerHTML* assignment, "*<s>*" tag will be rendered.

**Affected file:**
*web-rc/src/connectors/sso.ts*

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

**Affected code:**

```
function initiateBrowserSso(code: string, state: string) {
    window.postMessage({ command: 'authResult', code: code, state: state },
'*');
    const handOffMessage = ('; ' + document.cookie).split('; 
ssoHandOffMessage=').pop().split(';').shift();
    document.cookie = 'ssoHandOffMessage=;SameSite=strict;max-age=0';
    document.getElementById('content').innerHTML =
        `<p>${handOffMessage}</p>`;
}
```

It is recommended to display the string as plain-text by utilizing the *innerText* DOM property instead of *innerHTML*.

### BWN-02-013 WP1: XPath injection in SAML library *(Info)*

It was found that the SAML library *Sustainsys* suffers from an XPath injection within the Signature Verification. The *GetIdElement* function is overridden by the library's logic which does not verify if the passed ID parameter is a valid *NCName*[3] before proceeding to embed it directly into an XPath query. This could introduce the risk of XML signature flaws in the library, particularly when identifying the element referred to by the XML signature. However, as all relevant ID attributes are usually signed, this issue is rated as purely informational.

**Affected file:**
https://github.com/Sustainsys/Saml2/blob/develop/Sustainsys.Saml2/XmlHelpers.cs#L37

**Affected code:**

```
   public override XmlElement GetIdElement(XmlDocument document, string id)
    {
        var nodes = document.SelectNodes(
            $"//*[name() != 'Reference' and (@id='{id}' or @iD='{id}' or
              @Id='{id}' or @ID='{id}')]");
        [...]
        return (XmlElement)nodes[0];
```

It is recommended to report this issue to the library maintainers and wait for a fix, ideally validating the *id* argument before using it in the XPath query.

[3] https://www.w3.org/TR/1999/REC-xml-names-19990114/#NT-NCName

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-014 WP3: Insecure web preferences for Electron renderer *(Info)*

It was discovered that certain Electron's security-related options or features are not used properly in the desktop application. The current settings could lead to RCE if an attacker could find a way to execute arbitrary JavaScript on the renderer (e.g. via XSS). The recommended settings are listed next.

- **Disable Node.js integration**: If this is not disabled, an attacker can use any Node.js feature just by using the *require()* function and achieve RCE via that call. To disable this, set the *nodeIntegration* property to *false* in the *BrowserWindow* constructor's argument.
- **Enable context isolation**[4]: If this is not enabled, a web page's JavaScript can affect the execution of the Electron's internal JavaScript code on the renderer and the preload scripts. Since the Electron's internal code and *preload* scripts have access to Node.js features, in the worst-case scenario, an attacker can perform RCE by accessing powerful features via a specifically crafted JavaScript code on the web page. To enable this, set the *contextIsolation* property to *true* in the *BrowserWindow* constructor's argument.
- **Enable sandbox**[5]: This mitigates the harm that malicious code can cause by limiting access to most system resources. This is important to hinder an attacker's possibilities when the renderer has been compromised. Without the sandbox, arbitrary code execution can be achieved through publicly known Chromium bugs when an attacker can execute arbitrary JavaScript inside the renderer. To enable the sandbox for all renderers, call the *app.enableSandbox()* API before the *app*'s *ready* event is emitted.

Currently, these recommended settings are not used. The affected code was found in the following file and can be consulted next.

**Affected file:**
*jslib-rc/electron/src/window.main.ts*

**Affected code:**

```
async createWindow(): Promise<void> {
    [...]
    this.win = new BrowserWindow({
        [...]
        webPreferences: {
            nodeIntegration: true,
            backgroundThrottling: false,
            contextIsolation: false,
```

[4] https://www.electronjs.org/docs/latest/tutorial/context-isolation
[5] https://www.electronjs.org/docs/latest/tutorial/sandbox

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
        },
    });
    [...]
}
```

Although no way to execute arbitrary JavaScript was found in this test, it is recommended to use the aforementioned security options properly to remove the potential RCE risks. The proposed fix is included next, yet it must be noted that Bitwarden is aware of this issue. The in-house team is working on adapting the Electron application so the proposed fix can be deployed without breaking functionality.

**Proposed fix:**

```
app.enableSandbox();
[...]
async createWindow(): Promise<void> {
    [...]
    this.win = new BrowserWindow({
        [...]
        webPreferences: {
            nodeIntegration: false,
            backgroundThrottling: false,
            contextIsolation: true,
        },
    });
    [...]
}
```

### BWN-02-015 WP1: Insufficient subject confirmation in SAML library *(Info)*

It was found that the subject confirmation data of SAML assertions are not verified by the underlying SAML library *Sustainsys*. This means the risk of attackers supplying crafted and unsigned SAML response messages that contain signed SAML assertions while responding to completely unrelated SAML authentication requests. Although this issue cannot be exploited in common scenarios, it is included for the sake of completeness.

**Affected file:**
*Sustainsys.Saml2/SAML2P/Saml2PSecurityTokenHandler.cs*

**Affected code:**

```
protected override void ValidateSubject(Saml2SecurityToken samlToken,
TokenValidationParameters validationParameters)
{
    base.ValidateSubject(samlToken, validationParameters);

    if(!samlToken.Assertion.Subject.SubjectConfirmations.Any())
    {
```

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
            throw new Saml2ResponseFailedValidationException(
                  "No subject confirmation method found.");
        }

        if(!samlToken.Assertion.Subject.SubjectConfirmations.Any(sc =>
           sc.Method == bearerUri))
        {
            throw new Saml2ResponseFailedValidationException("Only assertions with
subject confirmation method \"bearer\" are supported.");
        }
}
```

It is recommended to report this issue to the library maintainer and wait for a fix that ideally verifies all attributes of the *SubjectConfirmationData,* including the *InResponseTo* attribute.

### BWN-02-019 Web: General HTTP security headers missing *(Medium)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was found that the Bitwarden platform is missing certain HTTP security headers in HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent flaws linked to headers.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[6]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- Note that the CSP framework offers similar protection to *X-Frame-Options* in ways that overcome some of the shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers at the same time, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header as well.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on

[6] https://cure53.de/xfo-clickjacking.pdf

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

the XSS auditor itself with false-positives, e.g. Universal XSS[7] and similar. It is recommended to set the value to either **0** or **1; mode=block**. Note that most modern browsers have stopped supporting XSS filters in general, so this header is only relevant in case older browsers are supported by the web application in scope.

- **Strict-Transport-Security**: Without the HSTS header, a MitM could attempt to perform channel downgrade attacks using readily available tools such as *sslstrip*[8]. In this scenario the attacker would proxy clear-text traffic to the victim-user and establish an SSL connection with the targeted website, stripping all cookie security flags if needed. It is recommended to set up the header as follows:

  *Strict-Transport-Security: max-age=31536000; includeSubDomains;*

  Note that the HSTS *preload* flag has been left out as it is considered dangerous[9].

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the aforementioned headers to every server response, including error responses like *4xx* items. More broadly, it is recommended to reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

### BWN-02-020 Web: Additional hardening of deployed CSP rules *(Info)*

The Bitwarden web vault makes use of the security offered by the *Content-Security-Policy* response header. As every permitted domain or set keyword can weaken or even render this security header useless, it has to be configured as strictly as possible. As shown below, the *style-src* directive contains the '*unsafe-inline*' property, which allows loading arbitrary CSS via in-line style tags. In case the web vault suffers from an HTML injection vulnerability, an attacker could abuse this behavior to leak secret information stored in a user's web vault.

**Deployed CSP:**

```
default-src 'self'
script-src 'self' 'sha256-ryoU+5+IUZTuUyTElqkrQGBJXr1brEv6r2CA62WUw8w=' https://
js.stripe.com https://js.braintreegateway.com https://www.paypalobjects.com
```

[7] http://www.slideshare.net/masatokinugawa/xxn-en
[8] https://github.com/moxie0/sslstrip
[9] https://www.tunetheweb.com/blog/dangerous-web-security-features/

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
style-src 'self' 'unsafe-inline' https://assets.braintreegateway.com
https://*.paypal.com
img-src 'self' data: https://icons.bitwarden.net https://*.paypal.com
https://www.paypalobjects.com https://q.stripe.com https://haveibeenpwned.com
https://www.gravatar.com
child-src 'self' https://js.stripe.com https://assets.braintreegateway.com
https://*.paypal.com https://*.duosecurity.com https://*.duofederal.com
frame-src 'self' https://js.stripe.com https://assets.braintreegateway.com
https://*.paypal.com https://*.duosecurity.com https://*.duofederal.com
connect-src 'self' wss://notifications.bitwarden.com
https://notifications.bitwarden.com https://cdn.bitwarden.net
https://api.pwnedpasswords.com https://2fa.directory/api/v2/totp.json
https://api.stripe.com https://www.paypal.com https://api.braintreegateway.com
https://client-analytics.braintreegateway.com https://*.braintree-api.com
https://bitwardenxx5keu3w.blob.core.windows.net
object-src 'self' blob:
```

It should be taken into consideration to remove the *'unsafe-inline'* property and instead protect stylesheets in a manner similar to how the *script-src* directive is already configured. This would form an additional hurdle for an attacker who wishes to exploit a HTML injection vulnerability.

### BWN-02-021 Web: Cross-Origin-related HTTP security headers missing *(Info)*

It was found that the Bitwarden platform is missing several of the newer[10] Cross-Origin-infoleak-related HTTP security headers in its responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems, such as for example issues relating to the Spectre attack[11]. The following list enumerates the headers that need to be reviewed to prevent flaws linked to headers.

- **Cross-Origin Resource Policy (CORP)** and *Fetch Metadata Request* headers allow developers to control which sites can embed their resources, such as images or scripts. They prevent data from being delivered to an attacker-controlled browser-renderer process, as seen in *resourcepolicy.fyi* and *web.dev/fetch-metadata*.
- **Cross-Origin Opener Policy (COOP)** lets developers ensure that their application window will not receive unexpected interactions from other websites, allowing the browser to isolate it in its own process. This adds an important process-level protection, particularly in browsers which do not enable full Site Isolation; see *web.dev/coop-coep.*
- **Cross-Origin Embedder Policy (COEP)** ensures that any authenticated resources requested by the application have explicitly opted-in to being loaded.

[10] https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html
[11] https://meltdownattack.com/

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Today, to guarantee process-level isolation for highly sensitive applications in Chrome or Firefox, applications must enable both COEP and COOP; see *web.dev/coop-coep.*

Overall, missing Cross-Origin security headers can be considered a bad practice that should be avoided in times where attacks such as Spectre are known to be well-exploitable and exploit code is publicly available. It is recommended to add the aforementioned headers to every relevant server response. Resources explaining those headers are available online, explaining both the proper header setup[12] as well as the possible consequences of not setting them after all[13].

### BWN-02-022 WP1: Potential tag expression injection in NotificationHub *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

It was found that the *POST /send* endpoint of the push controller receives the user ID from the request body as a string before passing it to the push notification service. For the NotificationHub implementation, the user ID is embedded unsanitized into the tag expression. This means the risk of attackers injecting and modifying the tag expression syntax in order to send a notification to all organizations or all users available on the platform.

This could be potentially abused by attackers to spam or desynchronize the state of end-users' devices and the Bitwarden API. However, it could not be verified or exploited during the engagement, due to a bug when authenticating with an installation key. Therefore, this issue is rated as purely informational.

**Affected file:**
*src/Core/Services/Implementations/NotificationHubPushNotificationService.cs*

**Affected code:**

```
public async Task SendPayloadToUserAsync(string userId, PushType type, object
payload, string identifier,
        string deviceId = null)
{
        var tag = BuildTag($"template:payload_userId:{userId}", identifier);
        await SendPayloadAsync(tag, type, payload);
        if (InstallationDeviceEntity.IsInstallationDeviceId(deviceId))
        {
        await _installationDeviceRepository.UpsertAsync(new
                InstallationDeviceEntity(deviceId));
```

[12] https://scotthelme.co.uk/coop-and-coep/
[13] https://web.dev/coop-coep/

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
        }
}
```

It is advisable to validate that all embedded strings only contain allowed characters from a pre-selected minimal character-set which forbids the use of special characters relevant for the tag expression. By doing so, attackers can no longer use those characters to escape from the tag within the expression, mitigating this vulnerability.

### BWN-02-023 WP1: Multiple time-unsafe string comparison of secrets *(Info)*

***Note:*** *The issue has been addressed and fixed during the assessment.*

Several occasions were found where the Bitwarden server application made use of time-unsafe string comparison, for instance when comparing the authentication secret supplied to the several WebHook billing controllers. This induces the risk of attackers abusing the linear relationship between the runtime of the comparison and the equivalent prefix-strings of the operands. Attackers could use it to accumulate and measure the exact runtime of requests, allowing extraction of the token character by character. As this attack is usually extremely difficult to perform, this issue has been downgraded.

**Affected file:**
*src/Billing/Controllers/StripeController.cs*

**Affected code:**

```
[HttpPost("webhook")]
public async Task<IActionResult> PostWebhook([FromQuery] string key)
{
        if (key != _billingSettings.StripeWebhookKey)
        {
        return new BadRequestResult();
        }
```

**Also affected:**
*src/Billing/Controllers/*Controller.cs*
*src/Api/Utilities/ApiHelpers.cs:HandleAzureEvents()*
*src/Core/Services/Implementations/UserService.cs:RecoverTwoFactorAsync()*

It is recommended to use a time-constant string comparison when comparing user-supplied data against secrets. By doing so, the relationship between the runtime of the function does not exist and cannot be abused by attackers to optimize the brute force attacks against the secret.

**Dr.-Ing. Mario Heiderich, Cure53**
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

### BWN-02-025 WP1: Potential SSRF via SAML *ArtifactBinding* (*Info*)

It was found that the SAML library offers support for *ArtifactBindin*g, eventually allowing attackers to perform a blind server-side request to an arbitrary web URL. Attackers could eventually access services within the internal network - like the Redis cache. Due to an unknown bug and the time-limitations of this audit, this issue could not be confirmed and is included as purely informational.

**Affected file:**
*Sustainsys.Saml2/WebSSO/Saml2ArtifactBinding.cs*

**Affected code:**

```
private static XmlElement ResolveArtifact(
       string artifact,
       StoredRequestState storedRequestState,
       IOptions options)
{
       var binaryArtifact = Convert.FromBase64String(artifact);
       var idp = GetIdp(binaryArtifact, storedRequestState, options);
       var arsIndex = (binaryArtifact[2] << 8) | binaryArtifact[3];
       var arsUri = idp.ArtifactResolutionServiceUrls[arsIndex];

       [....]

       var response = Saml2SoapBinding.SendSoapRequest(payload, arsUri,
clientCertificates);
```

It is recommended to use a different SAML library that allows disabling of the artifact binding. In effect, attackers will not be able abuse this feature to access internal services.